Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Nihad Kerić

Razvoj spletne aplikacije za nadzor vnosa hranilnih snovi pri bolnikih v enoti intenzivne nege

diplomsko delo

univerzitetni študijski program prve stopnje
računalništvo in informatika

Mentor: doc. dr. Aleš Smrdel

Ljubljana, 2017

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Pri bolnikih, ki so hospitalizirani v enoti intenzivne nege, je eden izmed dejavnikov, ki pripomorejo pri boljšem okrevanju in posledično hitrejši odpustitvi iz enote, tudi pravilna in zadostna prehrana. Cilj diplomske naloge je razvoj spletne aplikacije za spremljanje vnosa hranilnih snovi za posameznega bolnika po dnevih in zaužiti hrani. V okviru tega cilja je vaša naloga, da razvijete podatkovni model, ki bo omogočal spremljanje želenega in dejanskega vnosa hranilnih snovi za posameznega bolnika po dnevih, in ga prenesete na podatkovno bazo. Z uporabo primernega razvojnega okolja pa razvijte sistem, ki omogoča vnos in popravljanje informacij za posameznega bolnika v razvito podatkovno bazo, vnos in popravljanje posamezne hrane ter njenih hranilnih vrednosti, vnos prejete hrane za posamezen dan za bolnika glede na tip hrane ter izračun prejetih hranilnih snovi. Za lažje spremljanje prehrane bolnika pa implementirajte tudi izris prejetih hranilnih snovi po dnevih glede na zahtevan dnevni vnos. Implementirajte tudi uporabnike in omogočite določitev pravic, ki jih ima za podatkovno bazo, za vsakega uporabnika.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Nihad Kerić sem avtor diplomskega dela z naslovom:

Razvoj spletne aplikacije za nadzorvnosa hranilnih snovi pri bolnikih v enoti intenzivne nege (angl. *Development of web application for the control of nutrient intake in patients in intensive care unit*)

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Aleš Smrdela,
- so elektronska oblika diplomskega dela, Razvoj spletne aplikacije za nadzor vnosa hranilnih snovi pri pacientih v enoti intenzivne nege (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 3. januarja 2017 Podpis avtorja:

Rad bi se zahvalil mentorju doc. dr. Alešu Smrdelu za vso strokovno pomoč ter nasvete pri načrtovanju in izvedbi diplomske naloge. Posebna zahvala pa gre mojima staršema Senadu in Kasimu ter sestri Nermini za moralno podporo in strpnost ob izdelavi diplomske naloge.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
HTML	HyperText Markup Language	Označevalni jezik za izdelavo spletnih strani
CSS	Cascading Style Sheets	Kaskadne slogovne predloge
PHP	PHP Hypertext Preprocessor	Programski jezik PHP
SQL	Structured Query Language	Strukturiran poizvedbeni jezik
MVC	Model-View-Controller	Arhitekturni vzorec (Model Pogled Krmilnik)
ORM	Object-relational mapping	Objektno-relacijsko mapiranje

Povzetek

Naslov: Razvoj spletne aplikacije za nadzor vnosa hranilnih snovi pri bolnikih v enoti intenzivne nege

Diplomska naloga obravnava zajem, shranjevanje in spremljanje vnosa hranilnih snovi pri bolnikih v enoti intenzivne nege. Osnovni cilj je bil načrtati dober podatkovni model in realizirati spletno aplikacijo za rokovanje s podatki. Aplikacija omogoča uporabniku z administratorskimi pravicami dodajanje, urejanje in brisanje uporabnikov ter določanje njihovih pravic. Uporabniki spletne aplikacije pa lahko potem, glede na dodeljene pravice, vnašajo novega bolnika in vse podatke povezane z njim ali pa samo vnašajo oziroma popravljajo podatke o prehrani za določenega bolnika. Podatki o prehrani bolnika se shranijo v podatkovno bazo za evidenco ter nadaljnjo obdelavo. Pri implementaciji odjemalskega dela aplikacije pa smo uporabili tudi predloge Bootstrap in *Google Material Design* za boljšo uporabniško izkušnjo.

Ključne besede: spletna aplikacija, hranilne snovi,enota intenzivne nega,relacijska podatkovna baza, Laravel, PHP.

Abstract

Title: The development of web application for the control of nutrient intakes to patients in the intensive care unit

The diploma thesis deals with collecting, storing and following the nutrient intakes to patients in the intensive care unit. The basic objective was to sketch a good data model and to realize the web application for handling the data. By means of administrator rights, the application enables adding, editing and deleting users as well as determining their rights. With regards to the granted rights, the users of the web application can enter a new patient and all the data, connected with him, or they can only enter or correct the data on nutrition for a certain patient. The data on nutrition of the patient are stored in the database for records and further processing. In implementation of the client part of the application we also used the templates of Bootstrap and Google Material Design for a better user experience.

Keywords: web application, nutrients, intensive care unit, relational database, Laravel, PHP.

Poglavje 1

Uvod

Zdrava in pravilna prehrana je eden izmed pogojev za zdrav življenjski slog. Za opravljanje osnovnih življenskih funkcij organizma in hitrejše zdravljenje ter tudi okrevanje po bolezni, operaciji ali poškodbi pa je pravilna in zadostna prehrana izjemnega pomena. Da bi dobili vse nujno potrebne hranilne in energetske snovi v optimalni količini, je potrebno hrano uživati uravnoteženo. V odleku intenzivne nege zaposleni poskušajo voditi plan o prehrani bolnikov, da bi jim omogočili hitrejše zdravljenje. Delo je težko, ker je potrebno obdelati veliko podatkov. Včasih se je vse zapisovalo na papir in arhiviralo v mape, ki so zasedle veliko prostora. Informacijska tehnologija je olajšala shranjevanje velike količine podatkov, ker se lahko zapisujejo na primer v baze. Aplikacija, ki sem jo razvil v okviru diplomske naloge, olajša shranjevanje in arhiviranje podatkov. Zaposleni z njeno pomočjo hitro in enostavno pridobijo in vnašajo podatke bolnikov. V prvem poglavlju diplomske naloge sem predstavil tehnologijo in na kratko opisal, kaj sem uporabil. V nadaljevanju sem opisal področije in delo zaposlenih v oddelku intenzivne nege ter v tretjem in četrtem poglavlju narisal podatkovni model in opisal postopek implementacije aplikacije. Na koncu sem v predzadnjem in zadnjem poglavlju še opisal delovanje aplikacije in podal zaključek.

Poglavje 2

Uporabljene tehnologije

V tem poglavju predstavljamo tehnologije na strani odjemalca in na strani strežnika, ki smo jih uporabili pri izdelavi spletne aplikacije.

2.1 HTML

Pred razvojem svetovnega spleta in njegovih sestavnih delov je bilo težko digitalno informacijo enostavno, hitro in strukturirano deliti med več ljudi. Z namenom, da bi rezultate raziskav lahko delili z drugim zaposlenimi v Evropski organizaciji za jedrske raziskave in da bi bili ti dostopni z obeh lokacij (Francije in Švice) je Tim Berners Lee leta 1989 predlagal hipertekstovni sistem kot rešitev tega problema. S tem se je začel razvoj jezika za opis povezanih dokumentov *HTML*, novembra 1991 pa je bil objavljen prvi opis dokumenta *HTML* [6]. *Hypertext Markup Language* (skrajšano *HTML*) je hipertekstovni (včasih tudi nadbesedilni) označevalni jezik za določanje strukture in vsebine dokumentov *HTML*. Te lahko sestavljajo različne komponente, kot so besedila, hiperpovezave, slike, video posnetki in druge vsebine. Dokumenti *HTML* so temelj svetovnega spleta (*ang. World Wide Web*) in jih prikazujejo spletni brskalniki. Poleg vsebin, ki jih je brskalnik prikaže, lahko vsebujejo dodatne informacije v obliki metapodatkov (npr. uporaba jezika v dokumentu, ime avtorja ali opis dokumenta).

World Wide Web Consortium (W3C) razvija in skrbi za standard *HTML*. Leta 2014 so izdali aktualno različico standarda *Hypertext Markup Language*, in sicer *HTML 5*.

Dokument *HTML* je sestavljen iz treh delov:

- *DOCTYPE* (deklaracija) se nahaja na samem začetku datoteke in določa različico standarda, v katerem je dokument zapisan in eventualno definicijo tipa dokumenta (DTD).
- Glava *HTML* (*head*) obsega predvsem tehnične in druge meta podatke, ki običajno niso prikazani v oknu brskalnika, vključujejo pa informacijo, pomembno za interpretacijo dokumenta (npr. nabor znakov) ter informacijo, potrebno za želen prikaz dokumenta (slogovne predloge).
- Telesa *HTML* (*body*) vsebuje podatke, zapisane z značkami *HTML*, ki se uporabljajo za pravilen prikaz samega dokumenta.

Primer strukture dokumenta *HTML* lahko vidimo v spodnji kodi, kjer je v prvi vrstici deklaracija standarda, sledi glava, čisto na koncu pa je telo dokumenta:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Naslov strani</title>
    <!-- Komentarji se ne prikazujejo v brskalniku-->
  </head>
  <body>
    <p>Vsebina</p>
  </body>
</html>
```

2.2 CSS

Cascading Style Sheets (skrajšano *CSS*) so kaskadne slogovne predloge, določene

v obliki preprostega slogovnega jezika, ki skrbi za izgled spletnih strani. Z njimi definiramo pravila, s katerimi določimo, kako se bodo posamezni elementi *HTML* dejansko prikazali na strani. Določamo lahko velikost, odmik, barve, obrobe, pozicije in še celo vrsto drugih lastnosti [6]. Lahko tudi nadziramo aktivnosti, ki jih uporabnik izvaja nad elementi strani, kot je na primer lebdenje (*ang. hover*) z miško nad nekim elementom *HTML*. Predloge so bile razvite z namenom konsistentnega načina podajanja informacij o slogu spletnih dokumentov ter z namenom ločitve dela programerjev in oblikovalcev spletnih strani. Bistvo *CSS* je ločitev strukture od izgleda strani, tako da poskrbimo za večjo preglednost dokumentov, temelječih na sintaksi *HTML*. Obenem omogočajo tudi uniformen izgled več strani na spletišču ali celo več spletišč.

Vse verzije *CSS*, od 1.0 do današnje aktualne verzije 3.0, uporabljajo isti temeljni koncept sintakse. Sintaksa je relativno enostavna in sestoji iz ključnih besed, ki se imenujejo slogovne lastnosti. Predlogo sestavlja množica pravil, določenih za enega ali več elementov *HTML*, ki jih izberemo z uporabo selektorjev. Ti določajo množico elementov v dokumentu *HTML*, nad katerimi bomo uporabili določene slogovne predloge. Pri določanju množice elementov lahko uporabimo različne selektorje:

- značka HTML - na primer **div**,
- razred - na primer **.naslov**,
- značka z razredom - na primer **div.naslov**,
- identifikator - na primer **#naslov**,
- značka z identifikatorjem - na primer **div#naslov**,
- psevdo razredi - na primer **a:link**.

Sintakso zapisa definicije sloga vidimo na spodnjem primeru:

selektor1 [, selektor2, selektor3 ...][:psevdo-razred ali ::psevdo-element] lastnost1: .....: vrednost1; [lastnost2: vrednost2; lastnost3: vrednost3; ...]

2.3 JavaScript

JavaScript je skriptni programski jezik, ki se uporablja pri izdelavi dinamičnih vsebin na spletnih straneh oziroma za dinamično spreminjanje spletnih strani na strani odjemalca. Sodi v skupino interpretiranih jezikov, pri katerih se programska koda ne prevede, ampak se po vrsti razčlenjuje, razčlenjeni ukazi pa se nato izvršijo [1].

JavaScript koda se lahko vključi neposredno v dokument *HTML*, lahko pa se v dokument *HTML* doda povezava na ustrezno datoteko s kodo, kar omogoča ločitev označevalne sintakse in programske kode in s tem tudi večjo berljivost. Je objektno usmerjen programski jezik in na prvi pogled je sintaksa podobna programskima jezikoma C# ter Java. Navkljub podobnemu imenu in podobni sintaksi pa ima ta z Javo bolj malo skupnega, saj se že pri dedovanju močno razlikuje. JavaScript uporablja prototipno dedovanje in ne dedovanje, temelječe na razredih, tako kot Java. S pomočjo JavaScripta je bilo v zadnjem času razvitih veliko uporabnih knjižnic; nekatere so prikazane na sliki 2.1.

Slika 2.1: Knjiznice, napisane v JavaScriptu

Tipična področja uporabe so:

- preverjanje veljavnosti vnešenih podatkov (validacija podatkov),
- prikaz pogovornega okna,
- pošiljanje asinhronih zahtevkov na strežnik (Ajax),

- branje in pisanje piškotkov,
- zaščita pred krajo in zlorabo elektronskih poštnih naslovov (Spam).

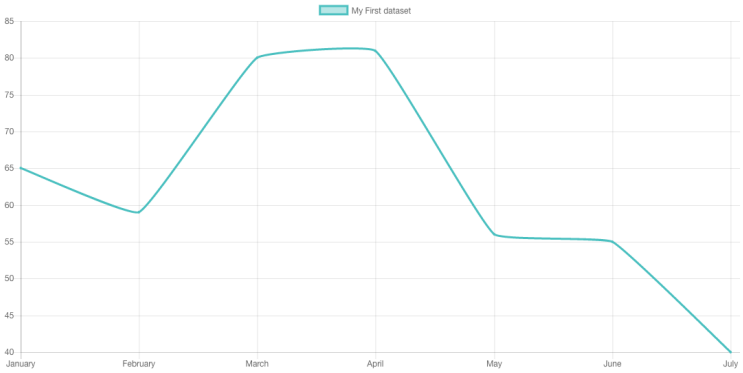
Slaba stran JavaScripta je v tem, da se lahko zlorabi za:

- neželena reklamna okna (Pop-up),
- neželeno zapiranje brskalniškega okna,
- spremembo velikosti brskalniškega okna,
- pobiranje slike ali teksta iz spletne strani,
- shranjevanje celotne spletne strani.

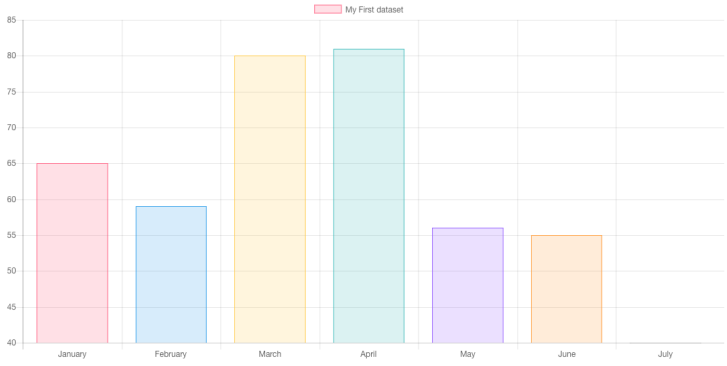
2.3.1 Chart.js

Chart.js je JavaScript knjižnica, ki omogoča, pripravo različnih vrst grafikonov s pomočjo komponente, definirane v standardu *HTML 5*, Canvasa [12]. Pri uporabi Canvasa je potrebno vključiti Polyfill za podporo starejšim brskalnikom. Knjižnica je neodvisna in je relativno majhna, saj njena stisnjena oblika obsega vsega 11 kB. Kljub relativni majhnosti pa se lahko uporabljena knjižnica še zmanjša, če v aplikaciji ne uporabljamo vseh šestih vrst grafov, ki jih knjižnica ponuja. S tem lahko malenkostno tudi prihranimo pasovno širino za uporabnike. Kot smo že omenili, knjižnica ponuja šest različnih grafov, in sicer:

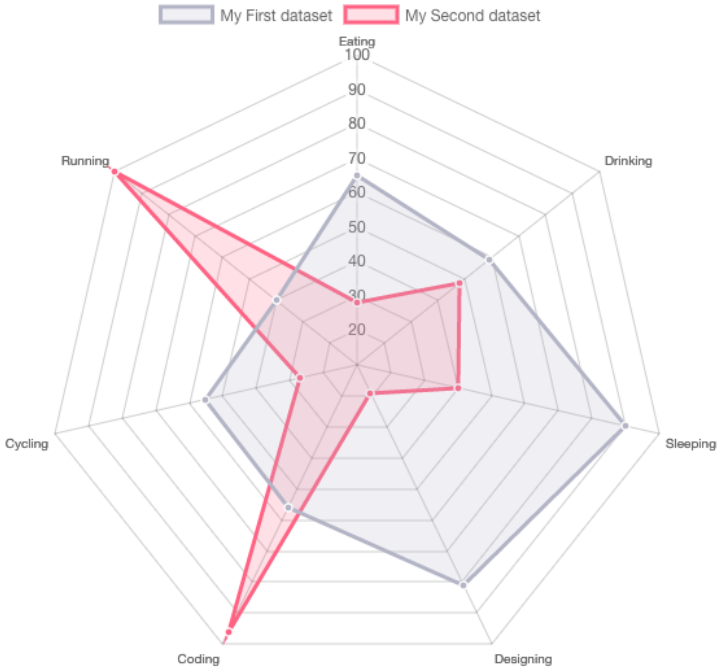
- črtni graf (*ang. Line Chart*), slika 2.2,
- stolpčni graf (*ang. Bar Chart*), slika 2.3,
- radar graf (*ang. Radar Chart*), slika 2.4,
- razrezani graf (*ang.Polar Area Chart*), slika 2.5,
- tortni graf (*ang. Pie & Doughnut Charts*), slika 2.6,
- mehurčni graf (*ang. Bubble Chart*) slika, 2.7.



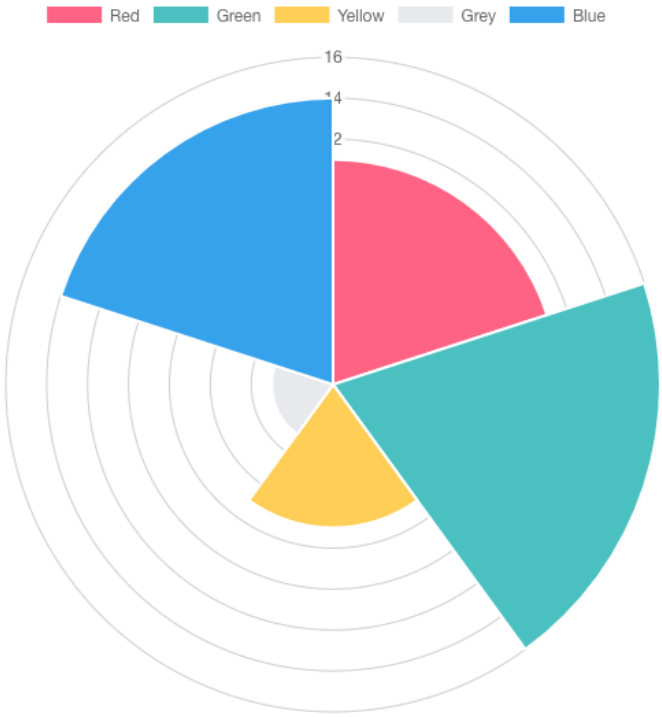
Slika 2.2: Črtni graf



Slika 2.3: Stolpčni graf

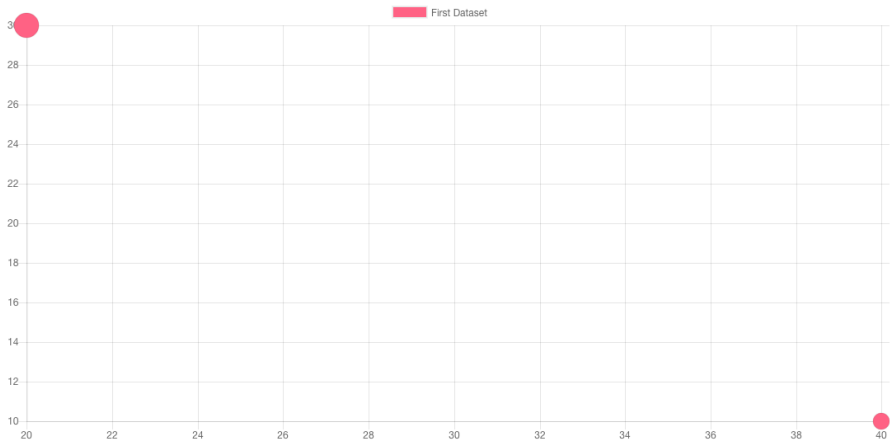


Slika 2.4: Radar graf



Slika 2.5: Razrezani graf

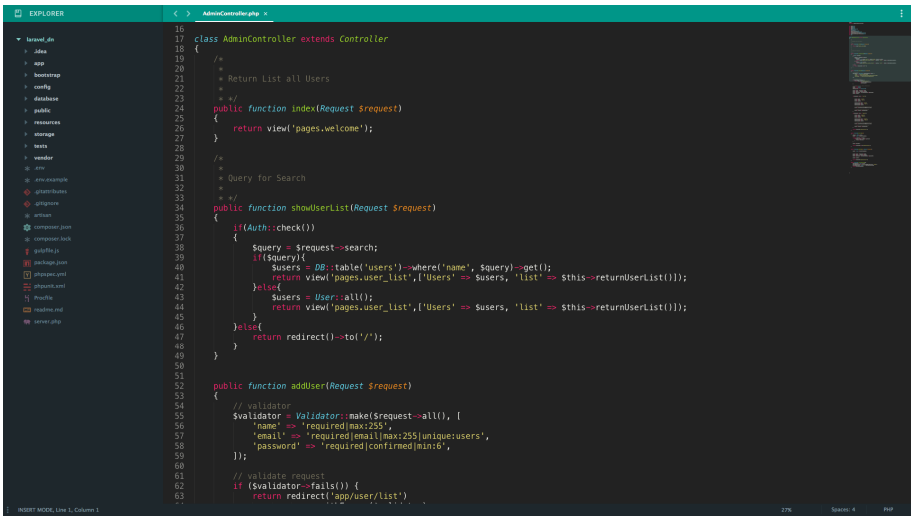
Slika 2.6: Tortni graf



Slika 2.7: Mehurčni graf

2.4 Laravel

Laravel je odprtokodno ogrodje za razvoj spletnih aplikacij v jeziku PHP [18], ki ga je razvil Taylor Otwell leta 2011. Taylor je razvil Laravel kot boljšo alternativo za *CodeIgniter*, saj slednji ni zagotavljal nekaterih funkcionalnosti, kot je na primer vgrajena podpora za avtentikacijo in avtorizacijo. Na sliki 2.8 je prikazano okno ogrodja Laravel s prikazano strukturo projekta na levi strani ter urejevalnikom kode z odprtim nadzornikom v preostalem delu okna.



Slika 2.8: Laravel s prikazano strukturo projekta na levi strani ter urejevalnikom kode z odprtim nadzornikom v preostalem delu okna

2.4.1 MVC

Model-View-Controller (MVC) je angleški izraz za vzorec za strukturiranje razvoja programske opreme. Nekateri avtorji, kot na primer [2], uvrščajo vzorec MVC v arhitekturni vzorec, medtem ko ga drugi uvrščajo v načrtovalni kot je to primer v [13]. Cilj vzorca je prilagodljiv programski model, ki omogoča lažjo kasnejšo spremembo ali razširitev. Po tem vzorcu se aplikacija razbije na tri dele: model, pogled in krmilnik. Vzorec omogoča tudi ponovno

uporabo posameznih komponent. Ena izmed dobrih lastnosti uporabe vzorca MVC je, da je z njegovo pomočjo mogoče narediti en model, do katerega lahko dostopa več aplikacij, ki so na različnih operacijskih sistemih, npr. Windows, Linux, Mac. Pri tem je model lahko povsem neodvisen, je pa potrebno za vsako aplikacijo za vsako platformo posebej napisati; tako krmilnik kot tudi pogled. Gre torej za arhitekturni oziroma načrtovalni vzorec, ki aplikacijo razdeli na tri med seboj odvisne komponente:

Model (*ang. Model*)

Model je skrbnik za podatkovno shemo, validacijo podatkov, konsistenčnost podatkov ter poslovno logiko. Akcije ali spremembe na pogledu, ki vplivajo na podatkovni model, potekajo preko krmilnika. Ko model spremeni podatke, krmilnik obvesti pogled, ta pa se ažurira.

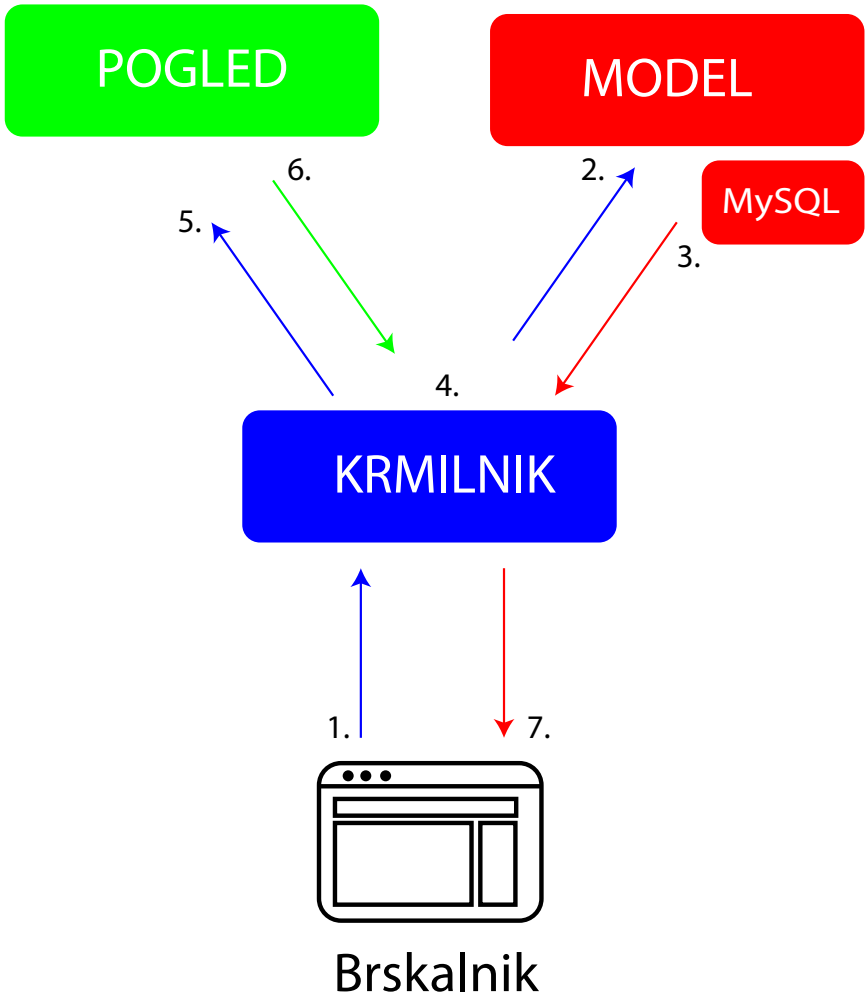
Krmilnik (*ang. Controller*)

V aplikacijski arhitekturi MVC ogrodja Laravel se krmilnik nahaja med predstavitveno plastjo in modelom. Skrbi za izmenjavo in obdelavo podatkov. Krmilnik upravlja z eno ali več predstavitvenimi plastmi. Intreakcija med krmilnikom in predstavitveno plastjo deluje v dveh smereh:

- Pogled pošlje podatke krmilniku, ki jih ustrezno obdela.
- Krmilnik odda podatke predstavitveni plasti in nato generirano stran predstavi končnemu uporabniku.

Pogled (ang. View)

Pogled je odgovoren za predstavitev zahtevanih podatkov iz modela in prejem uporabniške interakcije. Pogled oziroma predstavitvena plast je povezana s krmilinikom in modelom, ki podatke predstavlja, vendar ni odgovorna za nadaljnjo predelavo poslanih uporabniških podatkov.



Slika 2.9: Prikaz poteka interakcije pri arhitekturi Model-Pogled-Krmilnik

Laravel 5.2, ki je bil uporabljen pri razvoju aplikacije, predstavlja občutno izboljšavo predhodne različice, saj ima več različnih podpor [5] avtentikacijskih gonilnikov, možnosti implicitnega dodajanja modela, omogoča enostavnejšo uporabo na globalnih področjih, podpira vmesne (*ang. middleware*) skupine, izboljšuje pa tudi validacije nizov.

Interakcija v modelu MVC

Pri tem pristopu imamo tipično interakcijo med prej naštetimi komponentami. Slika 2.9 prikazuje tipične korake v interakciji spletne aplikacije, temelječe na arhitekturi MVC. Koraki, ki se izvajajo v tej interakciji, so:

1. Krmilnik prejema zahteve od brskalnika.
2. Krmilnik zahteva podakte od modela.
3. Model zahtevane podatke vrne krmilniku.
4. Krmilnik obdela podatke.
5. Pogled sprejema podatke.
6. Izdelan pogled se vrne krmilniku.
7. Izdelani pogled se pošlje brskanliku kot odgovor.

Poglavje 3

Opis področja

V tem poglavju sta predstavljena pomembnost zdravja in prehrane ter delo medicinske sestre v intenzivni negi (enota intenzivne terapije).

3.1 Pomebnost zdravja in prehrane

Dejstvo je, da je zdravje zelo odvisno od prehrane. Dejansko je tako, da sta zdravje in prehrana tesno povezana. Način prehranjevanja lahko deluje kot dejavnik tveganja za zdravje osebe, včasih pa tudi kot zaščitni dejavnik, ki pripomore k izboljšanju zdravja in kvalitete življenja. Številne raziskave [3, 11, 15, 16] so pokazale povezavo med pojavnostjo določenih bolezen in prehranjevalnimi navadami ter načinom življenja posameznika. Pri raziskavi [17] je bilo pokazano, da dejavniki nezdravega življenskega sloga sodijo med vzroke za nastanek bolezni, kot so: sladkorna bolezen, bolezeni srca in ožilja (kardio vaskularne bolezni), različne oblike raka. Na pojavnost fizioloških dejavnikov tveganja, kot so zvišane vrednosti holesterola in zvišan krvni tlak, vpliva tudi nepravilen (za holesterol je npr. ugotovljeno, da je le manjši del odvisen od prehrane, več je genetsko povezanega) način prehranjevanja. Da bi izboljšali kakovost življenja in preprečili nastanek številnih bolezni, je potrebno uravnotežiti prehrano. Primer piramide, ki prikazuje uravnoteženo in zdravo prehrano, je prikazan na sliki 3.1. Pri tem širina vsake vrstice

predstavlja primeren delež v prehrani.

Slika 3.1: Piramida zdrave prehrane

Uravnoteženje zdrave prehrane in pravilno prehranjevanje na splošno je izjemnega pomena za vse ljudi, še posebej pa je to pomembno pri bolnih osebah. Ena izmed kategorij bolnih oseb, ki so še posebej občutljive na razne dejavnike, so bolniki, ki se nahajajo v enoti intenzivne terapije. Pri njih je potrebno nadzorovati vnos različnih hranilnih snovi, kot so maščobe, ogljikovi hidrati in beljakovine, ter skupno energijsko vrednost. Vse to mora biti uravnoteženo, zadostno in tudi prilagojene bolnikovemu stanju (npr. bolniki z odpovedjo ledvic morajo zaužiti čim manj beljakovin). Uravnoteženost dosežemo tako, da zaužito hrano spremljamo in analiziramo ter po potrebi tudi korigiramo. To najlažje naredimo tako, da hrano razstavimo na različne hranilne snovi in načrtujemo ter nadziramo pravilen vnos posamezne hranilne snovi.

Pri prehrani ljudi je treba razlikovati med mikrohranili in makrohranili.

Mikrohranila so hranilne snovi, ki jih naše telo ne more proizvajati samo, in jih potrebujemo v manjši količini. V mikrohranila sodijo nekateri vitamini in mineralne snovi. Makrohranila pa naše telo uporablja kot gradivo ali gorivo, torej so to snovi, ki jih telo potrebuje v večjih količinah. V skupino makrohranil sodijo maščobe, beljakovine in ogljikovi hidrati.

3.2 Maščobe

Maščobe so pomembne za človeško telo [9], ker vsebujejo potrebne maščobne kisline, ki so:

- nasičene in
- nenasičene.

Maščobne kisline povečujejo nasitnost vrednosti hrane ter povečujejo energijsko gostoto hrane. Za vsakodnevno preživetje človek potrebuje vir osnovnih maščobnih kislin. S tem organizmu zagotovimo nujno potrebne maščobne kisline in medij za vitamine, topne v maščobah (A, D, E in K). Ljudje, ki vnašajo večjo količino maščob, so v veliki nevarnosti, da zbolijo za rakom ali boleznimi srca in ožilja. Maščobe v obroke vedno vključujemo skupaj z ogljikohidratnimi in beljakovinskimi živili.

3.3 Beljakovine

Organske molekule, ki jih lahko predstavimo kot verigo, sestavljeno iz manjših enot, imenujemo beljakovine. Te so zelo pomembne za človeško telo in so pomemben del naše prehrane. Telo samo proizvaja potrebne beljakovine, vendar moramo da je to sploh mogoče, vnašati hrano, ki ima beljakovine v sebi [8]. V procesu prebave se vnešene beljakovine razgradijo na aminokisline, ki jih telo nato uporabi za sestavljanje potrebnih beljakovin. Iz aminokislin so sestavljeni organi, tkiva in druge pomembne komponente organizma

Esencialne	**Nesencialne**
izolevcin	asparagin
levcin	alanin
metionin	cistein
lizin	aspartat
histidin	tirozin
arginin	serin
valin	prolin
tripofan	glicin
fenilalanin	glutamin
treonin	glutamat

Tabela 3.1: Vrste esencialnih in neesencialnih aminokislin.

[10]. Aminokisline delimo na dve vrsti, in sicer na esencialne in neesencialne. Seznam aminokislin se nahaja v tabeli 3.1.

Kot že ime samo pove, so najbolj potrebna vrsta aminokislin esencialne aminokisline. To so aminokisline, ki jih telo ne more sintetizirati samo in jih moramo zato nujno vnašati s hrano. Priporočljiv dnevni vnos beljakovin za odrasle osebe je v povprečju okoli 48 g na dan, vendar se s starostjo potrebe po beljakovinah nekoliko znižajo, a to zmanjšanje ni zelo izrazito. Če uživamo/zaužijemo večjo količino beljakovin, kot je priporočen dnevni vnos, lahko pride ob rednih prekoračitvah do različnih težav, kot so:

- Preveliko izločanje kalcija, kar ima negativne učinke na zdravje kosti.
- Osteoporoza - ki je bolezen, pri kateri je značilno postopno izgubljanje kostne mase.
- Ljudje, ki imajo zdravstvene težave z ledvicami, težje izločajo sečnino, ki je končni produkt presnove beljakovin.
- Visok nivo holesterola v krvi.

3.4 Ogljikovi hidrati

Ogljikovi hidrati se uporabljajo predvsem za hitro oskrbo z energijo v organizmu. Če je oskrba tkiv z ogljikovimi hidrati večja od porabe, se ostanek pretvori v maščobo in skladišči. Dobri ogljikovi hidrati so v živilih, ki niso industrijsko predelana. Takšna živila nas nasitijo za dalj časa in tako, da lahko naše telo pridobljeno energijo tudi porabi. Posledično se nam zaradi tega tudi ne nabirajo maščobne obloge oziroma se ne redimo.

Ogljikovi hidrati se delijo na:

- monodsaharide,
- disaharide in
- polisaharide.

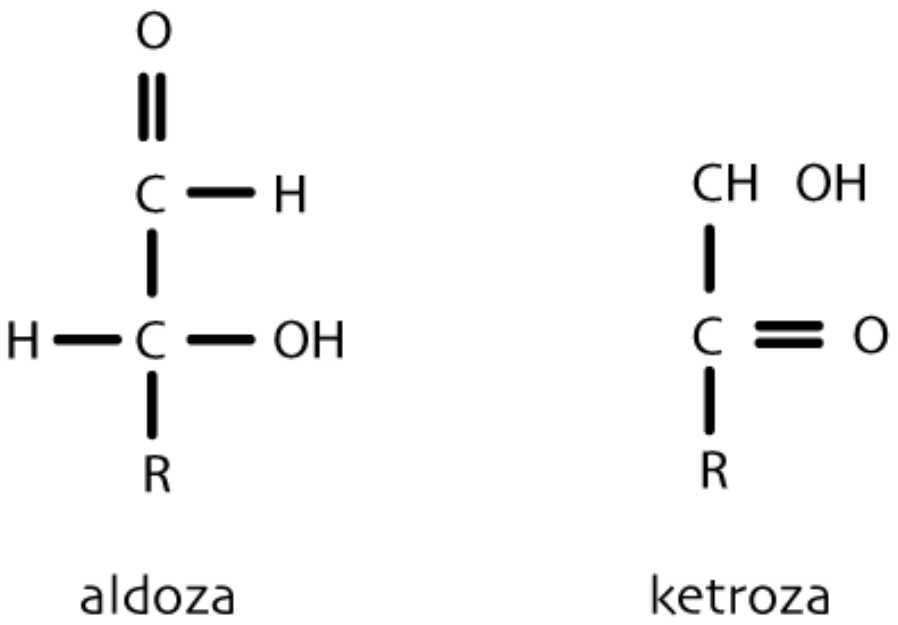
Monosaharidi

Monosaharidi (imenovani tudi enostavni sladkorji) so skupina organskih kemičnih spojin in gradniki vseh ogljikovih hidratov. Ne dajo se razstaviti na enostavnejše sladkorje. Vsi monosaharidi so sestavljeni iz verige vsaj treh ogljikovih atomov, kot temelj imajo karbonilno skupino in vsaj eno hidroksilno skupino. Monosaharide lahko delimo po številu ogljikovih atomov (tritoze, tetroze, pentoze) in po funkcionalnih skupinah (ketoza, aldoza). Najblj znana predstavnika monosaharidov pa sta glukoza in fruktoza. Na sliki 3.2 sta prikazani obe funkcionalni skupini monosaharidov.

Disaharidi

Disaharidi so sestavljeni iz dveh gradbenih enot monosaharidov. Nastajajo tudi kot produkt razgradnje polisharidov. Obstajajo tri vrste disaharidov, in sicer:

- saharoza (jedilni sladkor),
- maltoza (sladni sladkor),



Slika 3.2: Funkcionalni skupini monosaharidov.

- laktoza (mlečni sladkor).

Polisaharidi

Polisaharidi se lahko pridobijo na več načinov. Tako lahko na primer sestavimo več monosaharidov (najmanj tri), lahko pa tudi dve enoti disaharidov. Omenjena vrsta ogljikovih hidratov nima sladkega okusa in se ne topi v hladni vodi.

3.5 Vitamini

Za normalno delovanje telesa so potrebni vitamini. Naše telo ne more samo tvoriti vitaminov, zato jih je potrebno dobiti s hrano. Vitamine potrebujemo v majhnih količinah in zato sodijo med mikrohranila [4]. Vsak vitamin ima svojo vlogo, zato pomanjkanja enega vitamina ne moremo nadomestiti s povečanim uživanjem drugega. Vitamine lahko glede na topnost delimo na

vitamine, ki so:

- topni v vodi,
- topni v maščobah.

Brez osnovnih vitaminov ni mogoče vzdrževati življenja. Da dobimo dovolj vitaminov, moramo uživati veliko svežega sadja in zelenjave. V današnjem času poznamo veliko vitaminov, od katerih je trinajst takšnih, ki jih človeški organizem nujno potrebuje:

- vitamin A (retinol),
- vitamin C (askorbinska kislina),
- vitamin D (kalciferol),
- vitamin E (tokoferol),
- vitamin K (filokinon),
- biotin,
- folna kislina (vitamin B9 ali folacin ali folat),
- pantotenska kislina (vitamin B5 ali pantenol),
- kobalamin (vitamin B12),
- piridoksin (vitamin B6),
- niacin (vitamin B3 ali niacinamid ali nikotinska kislina ali nikotinamid),
- tiamin (vitamin B1),
- riboflavin (vitamin B2).

Pri neustrezni količini, bodisi premajhni oziroma preveliki, določenega vitamina se nam lahko zgodi, da pride do avitaminozo (popolno pomanjkanje), hipovitaminozo (delno pomanjkanje) ali hipervitaminozo (preveč vitaminov).

3.6 Delo medicinske sestre, odgovorne za hrano na oddelku za intenzivno nego

Na oddelku za intenzivno nego zdravijo bolnike, ki so življenjsko ogroženi. Da bi bilo zdravljenje na oddelku učinkovito, je potreben nadzor življenjskih funkcij, pa tudi prehrane. Z nadzorom življenjskih funkcij ponavadi ni večjih težav, se pa pogosto zaplete pri učinkovitem in natančnem nadzoru prehrane. Večina težav je povezana z zapisovanjem podatkov na listek, včasih pa se lahko zgodi, da ročno preračunavanje vnesenih hranilnih snovi ni vedno povsem pravilno. Pri listih se lahko med drugim zgodi tudi, da se izgubijo. V primerih, ko se izvaja prepis v elektronsko evidenco, pa lahko tudi pride do napake pri prepisovanju.

Da omilimo, če že ne odpravimo, zgoraj naštete težave, smo se odločili za razvoj spletne aplikacije, ki bi pomagala pri vodenju evidence o vnosu hranilnih snovi, ki jo bomo opisali v nadaljnjih poglavjih.

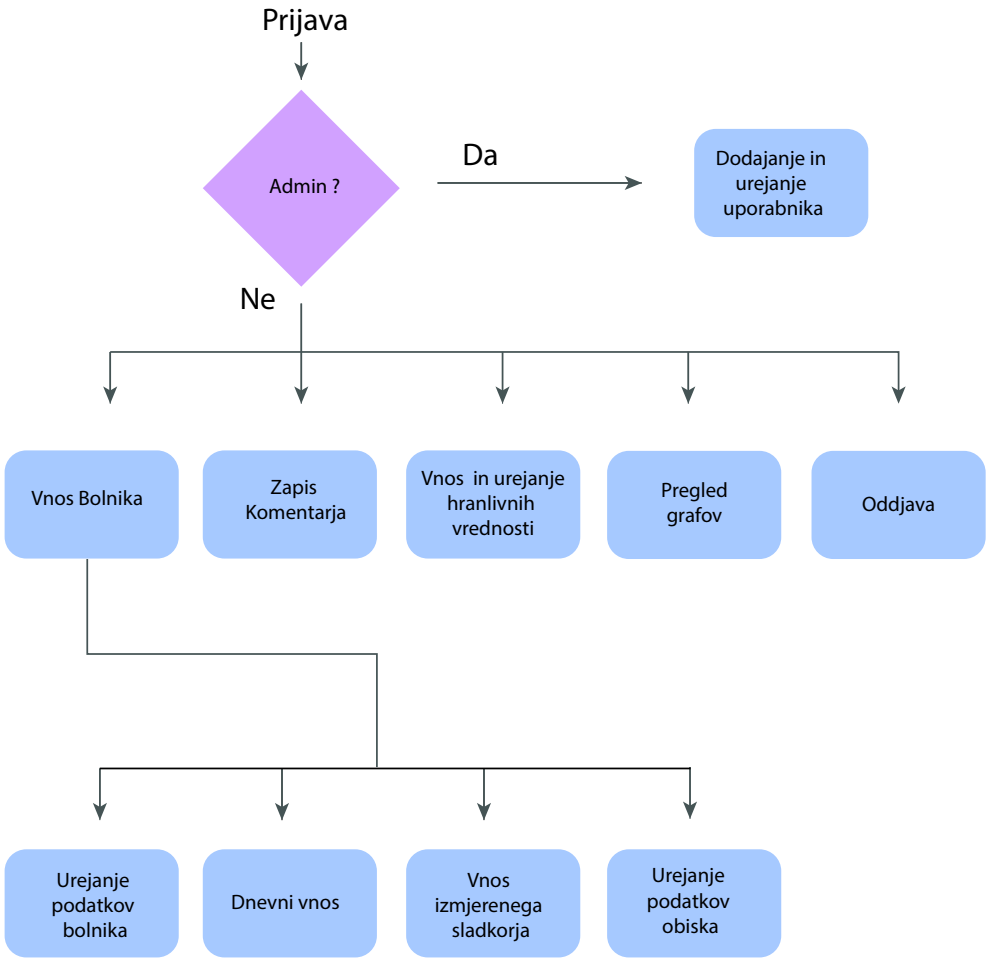
Poglavje 4

Razvoj podatkovnega modela

V tem poglavlju je predstavljen potek razvoja in načrtovanja podatkovnega modela. Namen aplikacije, ki je bila izdelana v okviru diplomske naloge, je pomoč zaposlenim (medicinskim sestram, ki se ukvarjajo s prehrano pacientov, nutricionistkam), da hitro in natančno beležijo in sproti preverjajo količino vnešenih hranljivih snovi bolnikov. Poleg tega pa lahko takšno zbiranje in prikaz podatkov služita tudi kot pripomoček pri nadaljnjem odločanju o terapiji.

4.1 Načrtovanje

Načrtovanje sem začel s raziskovanjem dela zaposlenih na oddelku intenzivne nege. Najprej je bilo potrebno ugotoviti, kje so težave in pri katerih aktivnostih se naredi največ napak. Ko sem pridobil vse potrebne podatke sem načrtovanje nadaljeval in izdelal diagram poteka, ki je prikazan na sliki 4.1. Diagram prikazuje funkcionalnosti, ki so na voljo uporabniku. Ta se mora seveda najprej prijaviti v aplikacijo. Nato aplikacija preveri katere vloge ima uporabnik. Privzeto so mu dodeljene le osnovne pravice, s katerimi lahko dostopa le do ožjega izbora funkcionalnosti. Če ima uporabnik tudi vlogo administratorja, pridobi možnost za dodajanje in urejanje podatkov vseh uporabnikov.



Slika 4.1: Diagram poteka

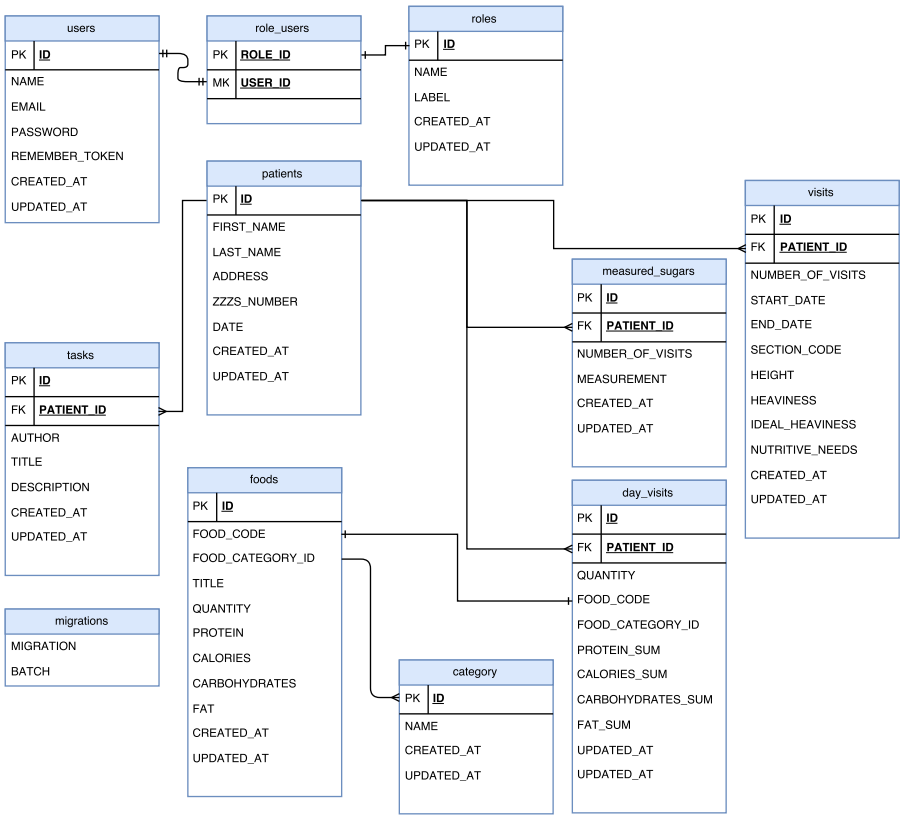
V naslednjem koraku sem se posvetil načrtovanju podatkove baze. V ta namen sem izdelal entitetno-relacijski (E-R) diagram, ki je prikazan na sliki 4.2.

Razviti E-R diagram vsebuje deset entitet s pripadajočimi atributi.

1. Entiteta *user* opisuje lastnosti uporabnika. Vsebuje osnovne atribute za prijavo uporabnika, kot so: *email, password, name.*

2. Entiteta *roles* opisuje vlogo uporabnikov.

3. Entiteta *role_users* povezuje entiti *user* in *roles.*



Slika 4.2: ER diagram

4. Entiteta *tasks*, služi shranjevanju komentarjev uporabnika.

5. Entiteta *foods* služi za shranjevanje različnih vrst hrane in hranljivih vrednosti.

6. Entiteta *patients* vsebuje osnovne podatke za posameznega bolnika.

7. Entiteta *measured_sugars* služi zapisovanju izmerjenih vrednosti sladkorja v krvi.

8. Entiteta *day_visits* služi za zapisovanje vnesenih vrednosti hrane za posameznega bolnika.

9. Entiteta *visits* služi za shranjevanje osnovnih informacij o obisku bolnika v enoti intenzivne terapije.

Poglavje 5

Implementacija

V tem poglavju je predstavljena implementacija aplikacije. Ogrodje Laravel ponuja zelo veliko pripomočkov, s čimer je močno olajšan razvoj spletnih aplikacij. Zelo pomemben pripomoček je *artisan* [14]. Gre za vmesnik v ukazni vrstici. Ponuja zelo veliko ukazov za hitro izdelavo različnih različnih komponent v okviru modela *MVC*. Ukazi, ki so najpomembnejši pri razvoju spletnih aplikacij in največ pripomorejo k hitremu in učinkovitemu razvoju, so:

- *php artisan migrate* ukaz, ki zažene preslikavo migracijskih razredov v tabele podatkovnih baz;
- *php artisan make:* ukaz, s katerim naredimo želen razred PHP, za dvopičjem navedemo eno od naštetih opcij (*migration, controller, model, seeder, provider*);
- *php artisan route:list* pregled vseh definiranih poti (*ang. route*);
- *php artisan serve* zažene integrirani razvojni strežnik PHP.

5.1 Ustvarjanje migracij in tabel v podatkovni bazi

Po načrtovanju diagrama E-R je bilo najprej potrebno s pomočjo vmesnika *artisan* narediti razrede, ki opisujejo tabele v podatkovni bazi (*Migration*). V ustvarjenih razredih sem opisal lastnosti entitet in njihove povezave. Na sliki 5.1 je prikazana izvorna koda migracijskega razreda za entiteto *users*. V razredu sta metodi *create* in *down*. V metodi *create*, sem definiral, kako se bodo lastnosti zapisale v podatkovno bazo. V nasprotju s tem pa se v metodi *down* definira, kako se bodo zbrisale lastnosti iz podatkovne baze. To je bilo potrebno narediti za vse entitete. Za tem je bilo potrebno pognati ukaz *php artisan migration*, ki v vsakem migracijskem razredu kliče metodo *create* in naredi tabele v podatkovni bazi. V primeru, ko bi želeli klicati metodo *down* in zbrisati vse tabele iz podatkovne baze, pa je potrebno uporabiti ukaz *php artisan migrate:rollback*.

```php
<?php

use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreateUsersTable extends Migration {

    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('users', function(Blueprint $table)
        {
            $table->increments('id');
            $table->string('name');
            $table->string('email')->unique();
            $table->string('password', 60);
            $table->rememberToken();
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::drop('users');
    }

}

```

Slika 5.1: Izvorna koda za migracijski razred entitete *users*.

5.2 Ustvarjanje modela

Laravel za dostop do podatkov v podatkovni bazi poleg možnosti povpraševanja SQL ponuja tudi dostop preko *Eloquent ORM*. Za hitrejse in lažje delo sem se odločil uporabiti *Eloquent ORM*, ki omogoča, da tabele v podatkovni bazi predstavljamo kot objekete, s katerimi lahko enostavno operiramo v programskem jeziku. Da bi uporabili *Eloquent ORM*, je potrebno za vsako tabelo v podatkovni bazi narediti model. Ta se zelo preprosto ustvari s pomočjo *ar-*

tisan vmesnika z ukazom *php artisan make:model NAZIV MODELA*. S tem ukazom se ustvari ustrezen model, ki ga lahko takoj uporabljamo za dostop do podatkovne baze, lahko pa tudi v modelu definiramo določene lastnosti za atribute tabele, kot so:

- ime tabele (*ang. table names*),
- primarni ključ (*ang. primary keys*),
- časovni žigi (*ang. timestamps*),
- povezava s podatkovno bazo (*ang. database connection*).

V modelu lahko tudi definiramo funkcije, v katerih opisujemo povezanosti med tabelami. Posebaj velja izpostaviti funkciji *hasRole* in *roles*. Funkcija *hasRole* preveri, če ima obstoječi uporabnik zapisano vlogo v tabeli *roles*, funkcija *roles* pa vrne vse vloge uporabnika. Vse omenjeno (lastnosti in funkcije) je prikazano na sliki 5.2 za model podatkovne tabele *user*.

V nadaljevanju je bilo potrebno narediti logiko v krmilnikih aplikacije. Za to je bilo potrebno najprej izvesti vse prej navedene korake za vse tabele v podatkovni bazi. Večino logike sem pisal v krmilniku, tako da so v modelu samo definirane povezave med tabelima, enako kot v modelu, namenjenemu hranjenju informacije bolnikov. V tem modelu sem definiral, ima več (*ang. hasMany*) povezavo s tabelami *day_visits*, *visits* in *measured_suggar*.

```php
class User extends Model implements AuthenticatableContract,
                                    AuthorizableContract,
                                    CanResetPasswordContract
{
    use Authenticatable, Authorizable, CanResetPassword;

    protected $table = 'users';
    protected $fillable = ['name', 'email', 'password'];
    protected $hidden = ['password', 'remember_token'];

    public  function roles()
    {
        return $this->belongsToMany(Role::class);
    }

    public function assignRole($role)
    {
        return $this->roles()->save(Role::whereName($role)->firstOrFail());
    }

    public function hasRole($role)
    {
        if(is_string($role))
        {
            return $this->roles->contains('name',$role);
        }

        return !! $role->intersect($this->roles)->count();
    }
}
```

Slika 5.2: Izvorna koda modela *users*

5.3 Izdelava krmilnikov in pisanje aplikacijske logike

Logika pri arhitekturi *MVC* je shranjena v krmilniku. Osnovo posameznih krmilnikov sem izdelal s pomočjo vmesnika *artisan*. Pri izdelavi aplikacije sem za vsako stran naredil svoj krmilnik.

5.3.1 Krmilnik za prijavo

Krmilnik *AuthController* je namenjen prijavi uporabnika v aplikacijo. Tudi pri prijavi Laravel olajša delo, ker ponuja PHP razred *AuthenticatesAndRegistersUsers*, ki vsebuje že narejeno logiko za preverjanje avtentikacije in avtorizacije uporabnika. Razred *AuthController* deduje lasnosti razreda

AuthenticatesAndRegistersUsers in s tem pridobi funkcije za prijavo, kot so: *getLogin()* in *postLogin()*. Le pri validaciji sem napisal svojo funkcijo *validator()*, v kateri preverim obliko naslova elektronske pošte in gesla. Pri naslovu elektronske pošte preverim, če je oblika pravilna (zahtevana oblika: primer@domena.si) in če niz ni daljši od 20 znakov. Pri preverjanju gesla pa se preveri le dolžina niza, ki ne sme biti krajša od 6 znakov.

5.3.2 Krmilnik za administracijo

V krmilniku *AdminController* je napisana logika za adminstracijo uporabnikov. Krmilnik vsebuje pet funkcij:

- Funkcija *index()* prikaže pogled za administracijo.
- Funkcija *showUserList()* najprej preveri, ali je uporabnik prijavljen in vrne/prikaže seznam vseh uporabnikov.
- Funkcija *addUser()* pridobi zahtevek (*ang. request*) in v prvem koraku validira podatke. Potem preveri, katero vlogo ima uporabnik ter to vlogo shrani v objekt *role*. Ostale podatke shrani v objekt *user* in vse objekte shrani v podatkovno bazo. V zadnjem koraku še poveže uporabnika z vlogo.
- Funkcija *destroy()* pridobi identifikacijo uporabnika ter poišče uporabnika v bazi in ga izbriše.
- Funkcija *update()* pridobi nekaj parametrov. Najprej mora pridobiti zahtevek (*ang. request*) in identifikacijo uporabnika (*ang. User ID*). Nato funkcija poišče določenega uporabnika in pridobljeni objekt shrani v spremenljivki. Potem spremeni podatke v objektu, shranjenem v spremenljivki. Objekt shrani nazaj v podatkovno bazo.

5.3.3 Krmilnik bolnika

V tem delu aplikacije sem krmilnike razdelil na dva dela, in sicer na:

- *PatientController* je namenjen izvajanju standardnih operacij: ustvarjanje, branje, posodabljanje in brisanje (*ang. Create, Read, Update, Delete*) bolnikov. Obstaja tudi dodatna funkcija, *changeDataFormat()*, s katero se spremeni ta obliki datuma in ure, preden se ju prikaže.
- *PatientDetailController* je krmilnik, v katerem je logika za upravljanje s tabelami (dnevni vnos, izmerjeni sladkor, obiski) za posameznega bolnika. V krmilniku je skupna funkcija *showDetails()*, ki prikaže podatke za vse tabele. Za vsako tabelo posebej je napisana programska logika za:
  - shranjevanje,
  - posodabljanje,
  - brisanje.

5.3.4 Krmilnik za hranilne vrednosti

V krmilniku *FoodController* je logika za pogled, ki se nahaja v datoteki *food.blade.php*. Krmilnik vsebuje naslednje funkcije:

- Funkcija *index()* prikaže pogled za hranilne snovi.
- Funkcija *store()* shrani novo vrsto hrane.
- Funkcija *update()* posodobi obstoječo hrano v podatkovni bazi.
- Funkcija *destroy()* briše obstoječo hrano iz podatkovne baze.
- Funkcija *makeFoodCode()* pridobi vnesene hranilne vrednosti, vrsto hrane in tromestno kode hrane, ter generira novo kodo hrane za podatkovno bazo. Ta funkcija najprej pretvori posamezno hranljivo vrednost v tri mestno celo število s funkcijo *converTo3Dig()*. V drugem koraku pogleda, v katero kategorijo hrane pripada vnesena hrana. V zadnjem koraku pa funkcija vse pridobljene podatke sestavi v en niz, ki ima obliko: (1 znak: vrsta hrane) + (3 znaki: proizvodna koda hrane) +

(3 znaki: količina maščob) + (3 znaki: količina beljakovin) + (3 znaki: količina ogljikovih hidratov) + (3 znaki: energijska vrednost).

5.3.5 Krmilnik za prikaz grafov

V Krmilniku *ChartController* se nahaja logika, ki vrne podatke za prikaz zahtevanih grafov. Krmilnk vsebuje funkcije:

- Funkcija *index()* vrne zahtevane podateke in pogled za prikaz grafov.
- Funkcija *convertCollectionToArray()* spremeni zbirko (*ang. Collection*) v niz (*ang. Array*) zaradi lažjega prikaza na X in Y osi grafa.

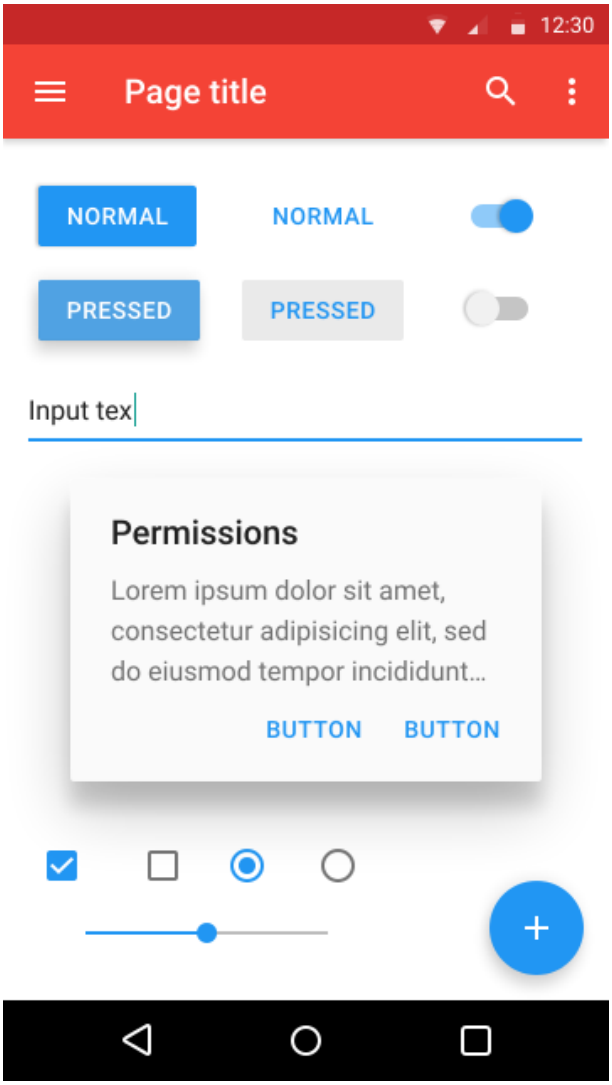
Poglavje 6

Opis delovanj aplikacije

V tem poglavju je predstavljen opis delovanja aplikacije in uporabniškega vmesnika. Vmesnik aplikacije je narejen v modernem Googlovem *Matrial Designu*.

6.1 Matrial Design

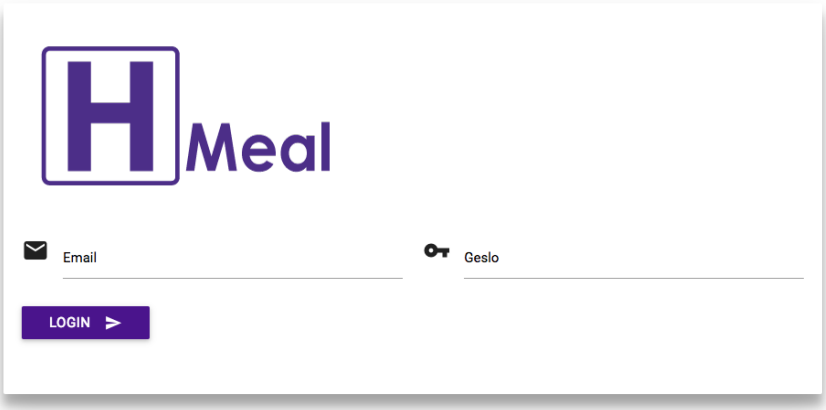
Material design (*ang. Material Desing*) je najnovejša smer Google dizajna, ki je narejen prvenstveno za Android OS in Googlove spletne aplikacije. Dizajn temelji na karticah podobnih površin in na ravnem oblikovanju (*ang. Flat Design*), ki je znan po svojem minimalizmu, vendar uporablja številne animacije in sence, ter tako ustvarja rahel učinek globine, obenem pa uporabnikom takoj razkrije, katera območja vsebujejo pomembno informacijo. Googlov oddelek za načrtovanje dizajna je razvil *Material Design* z upoštevanjem najboljše prakse (*ang. Best Practices*) ter interakcije uporabnika in spletne strani ali spletne aplikacije. Vsi gradniki (npr. gumb, besedilno polje, oznaka, spustni meni) so podrobno opisani. Elementi morajo temeljiti na oblikovalskih smernicah, vendar se lahko v izgledu med seboj razlikujejo [7]. Osnovni gradniki material dizajna so prikazani na sliki 6.1.



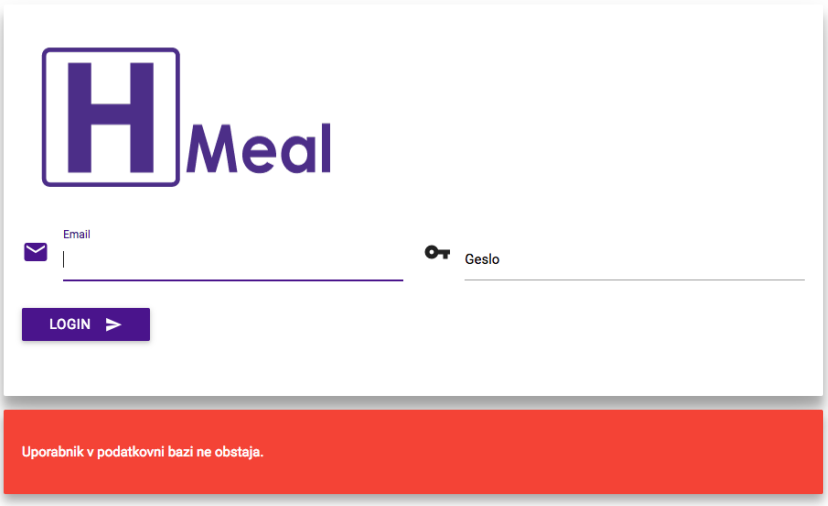
Slika 6.1: Prikaz osnovnih gradnikov Googlovega material dizajna *(ang. Material Design)*. Zgoraj so prikazani osnovi gradniki, kot so gumbi, potrditveno polje, obvestilno okno, besedilno polje in drugi najpogosteje uporabljani gradniki. Vidimo lahko tudi, kako minimalističen in preprost je dizajn. Vir: http://localsphere.com/a-brief-history-of-web-design/

6.2 Stran za prijavo

Stran za prijavo je zelo preprosta. Imamo samo en formular za vnos naslova elektronske pošte in gesla, kot je prikazano na sliki 6.2. Po pravilnem vnosu in kliku na gumb za prijavo (*ang. Login*) se podatki prenesejo v krmilnik za prijavo (*PatientController*). Če se zgodi, da vnos ni pravilen, se po kliku na gumb za prijavo izpiše obvestilo o napaki v rdečem pogovornem oknu. Stran za prijavo uporabnika v aplikacijo s prikazanim obvestilom o napaki je prikazana na sliki 6.3.



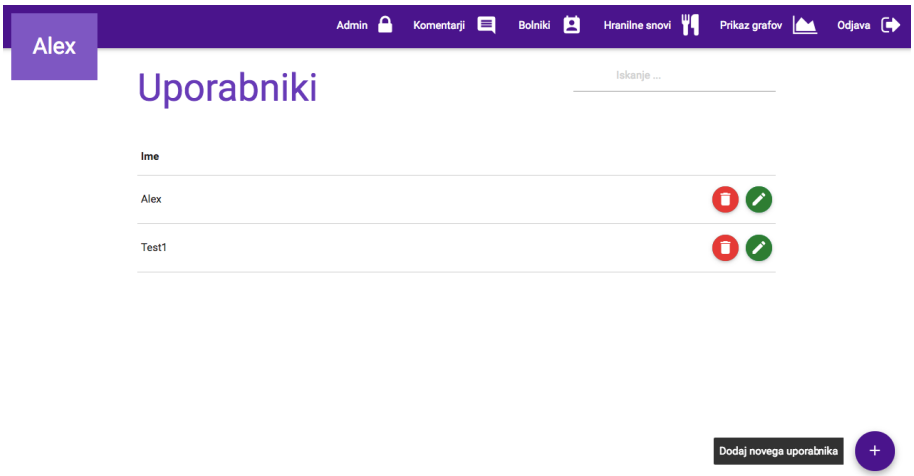
Slika 6.2: Stran za prijavo v aplikacijo



Slika 6.3: Stran za prijavo v aplikacijo s prikazano napako

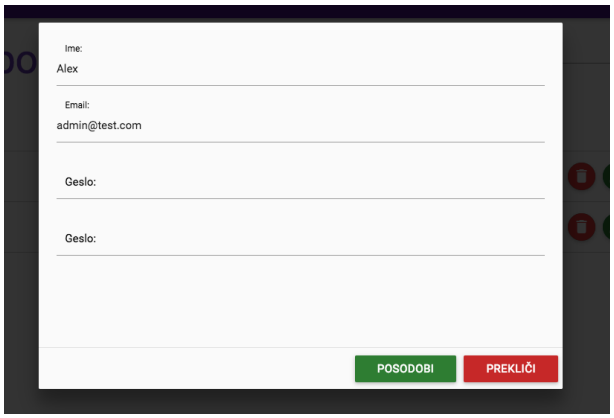
6.3 Administracijska stran

Če ima uporabnik aplikacije vlogo *admin*, se mu v navigaciji prikaže povezava do administracijske strani. Uporabnik lahko na njej pregleda vse vnesene uporabnike, ki so prikazani v tabeli. Slika 6.4 prikazuje osnovno administracijsko stran.

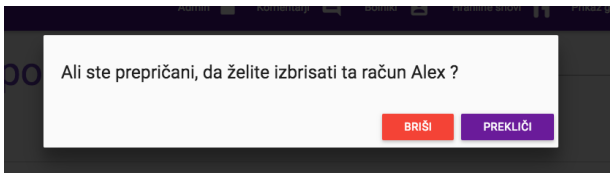


Slika 6.4: Stran za administracijo

V zgornjem desnem delu pod navigacijo je formular za iskanje posameznega uporabnika. Uporabniki so prikazani v glavnem delu okna, kjer je vsak prikazan v svoji vrstici. Poleg tega sta pri vsakem uporabniku dva okrogla gumba, ki omogočata urejanje oziroma brisanje uporabnika. Formular za urejanje podatkov o uporabniku je prikazan na sliki 6.5. S klikom na gumb za urejanje se prikaže formular, v katerem so zapisani podatki uporabnika. Formular za urejanje podatkov o uporabniku je prikazan na sliki 6.5. Če spremenimo podatke in ponovno pritisnemo na gumb za posodabljanje, se podatki uporabnika zamenjajo.



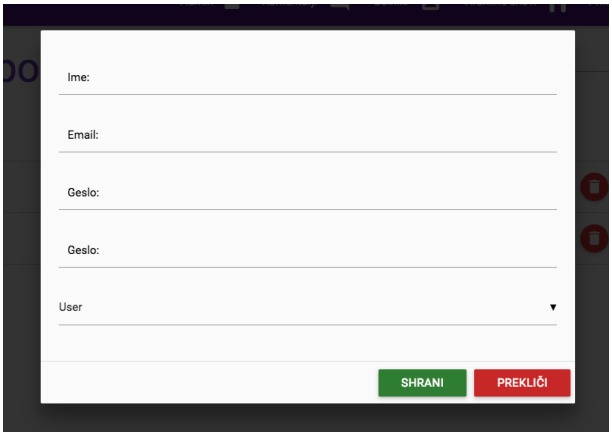
Slika 6.5:
Formular za urejanje uporabnika



Slika 6.6:
Vprašanje preden se uporabnika končno izbriše

Če želimo iz podatkovne baze izbrisati uporabnika, kliknemo na rdeči gumb s sliko koša za smeti (slika 6.6). Pri tem se pojavi formular, ki zahteva potrditev oziroma preklic akcije, kot je prikazano na sliki 6.6. Glavna aktiv-

nost pri administraciji uporabnikov se nahaja v spodnjem desnem kotu (slika 6.4). To je gumb za dodajanje novega uporabnika. Pri kliku na ta gumb se pojavi formular, ki je prikazan na sliki 6.7. Formular omogoča izpolnjevanje in shranjevanje podatkov o novem uporabniku.



Slika 6.7: Formular za dodajanje novega uporabnika

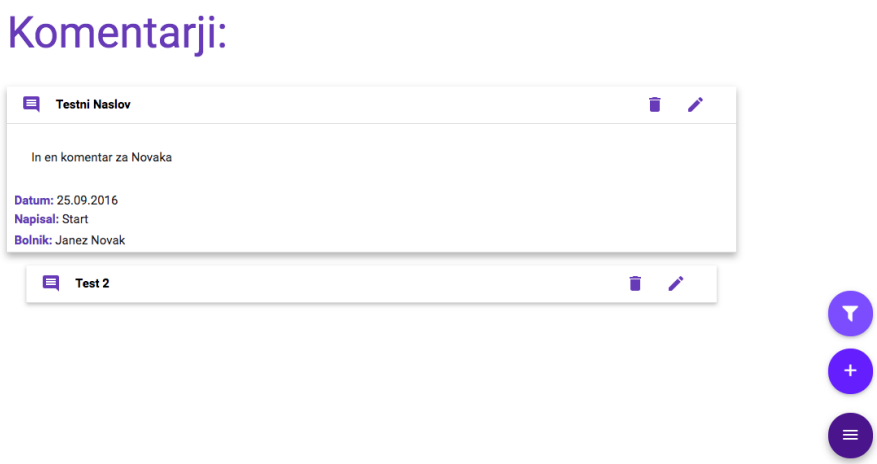
6.4 Stran za komentarje

V navigaciji je povezava "Komentarji", preko katere lahko pridemo na stran za komentarje, ki je prikazana na sliki 6.8. Na tej strani se nahaja seznam, ki vsebuje samo naslov tematike komentarja.

S klikom na ta naslov se pod tem komentarjem pokažejo (slika 6.8, zgornji komentar):

- komentar,
- datum in ura ter
- ime in priimek bolnika.

Komentar je možno tudi urejati preko akcij, dosegljivih z gumboma, ki se nahajata v glavi posameznega komentarja. Poleg tega ima uporabnik tudi

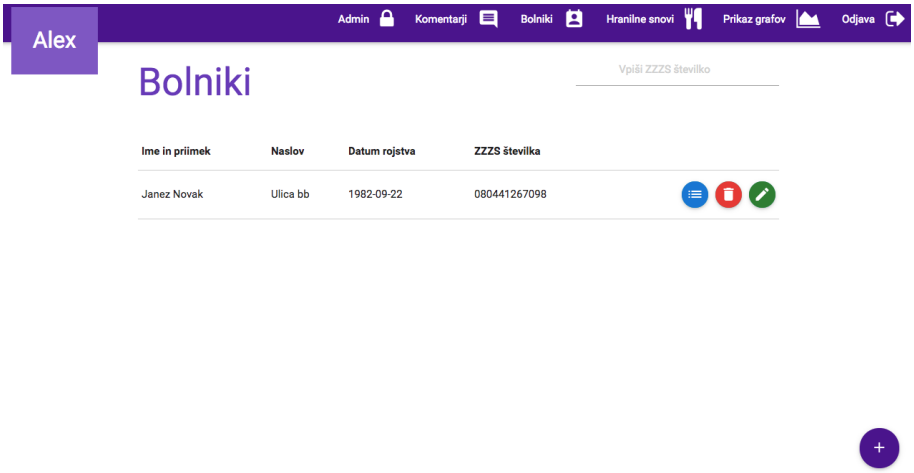


Slika 6.8: Stran za prikaz komentarjev

možnost detaljnega iskanja in dodajanja komentarja, ki je dosegljiva preko gumbov v spodnjem desnem delu strani.

6.5 Stran za bolnike

Na sliki 6.9 je prikazana stran bolnika, ki je podobna administracijski strani uporabnikov. Bolniki so prikazani v tabeli, kjer je vsak bolnik predstavljen v svoji vrstici, poleg pa se nahajajo gumbi, ki omogočajo urejanje in brisanje. V desnem spodnjem kotu se nahaja gumb za dodajanje novega bolnika.



Slika 6.9: Stran bolnikov

Modri gumb, ki je poleg gumbov za brisanje in urejanje pri vsakem bolniku posebej je gumb, ki nas prestavi na novo stran. Ta stran je namenjena urejanju vnosa hranilnih snovi. Stran je prikazana na sliki 6.10. Na novi strani so prikazane tabele v zavihkih za: dnevni vnos, izmjereni sladkor in obiske pacienta v enoti intenzivne nege. Za vsako omenjeno tabelo so implementirane operacije *CRUD* (*Create, read, update and delete*). Pri tabeli za izmerjen sladkor je tudi implementiran prikaz, ki opozori uporabnika v primeru, če ima bolnik preveliko meritev sladkorja. V primeru, ko je nivo sladkorja v krvi prevelik, se vnos obarva z rdečo barvo, če pa postavimo miško nad obarvano vrednost, pa se prikaže sporočilo "Pozor, možna sladkorna bolezen".

Slika 6.10: Stran za prikaz tabel bolnika

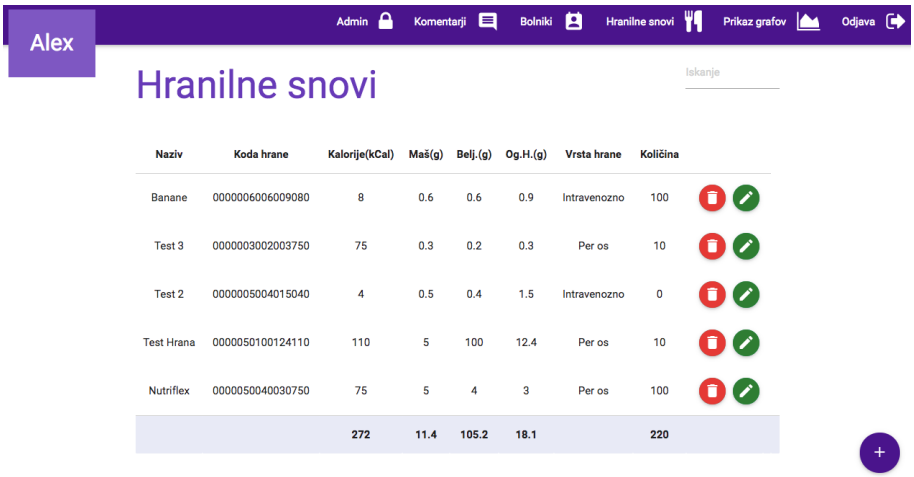
6.6 Stran za hranilne snovi

Strani za prikaz hranilnih snovi je podobna strani za bolnika. Prikazana je na sliki 6.11. Koncept razporeditve elementov je enak na vsaki strani. Tabela za prikaz vnosa hranil nih snovi prikazuje:

- naziv hrane,
- sestavljeno kodo hrane,
- količino kalorije v meritveni enoti (kCal),
- količino maščob v meritveni enoti (g),
- količino beljakovin v meritveni enoti (g),
- količino ogljikovih hidratov v meritveni enoti (g),
- vrsto hrane, ki lahko zavzame vrednosti *Intravenozno* ali *Per os*,
- količino hrane.

Hrana se lahko doda s pritiskom na gumb, ki se nahaja v spodnjem desnem kotu. V tabeli ima tudi vsaka vrstica posebna gumba, s katerima lahko

uporabnik prikliče strani, ki omogočajo urejanje oziroma brisanje hrane. Tabela omogoča tudi prikaz seštevka posameznih zaužitih hranilnih snavi oziroma celotne energijske vrednosti.



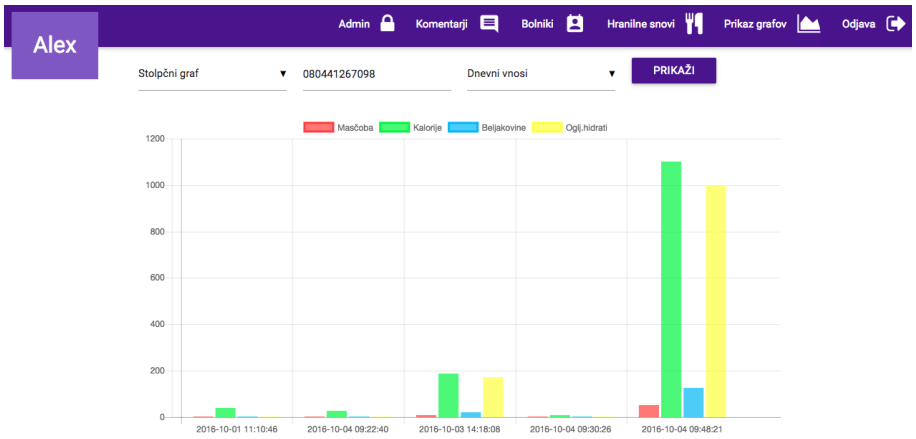
Slika 6.11: Stran za prikaz hranilnih snovi

6.7 Stran za prikaz grafov

Da bi lažje pregledali vnesene podatke za vse bolnike, sem implementiral tudi vizualni prikaz podatkov. Slika 6.12 prikazuje stran z grafičnim prikazom podatkov. Da bi se prikazal graf na strani, je potrebno vnesiti podateke:

- vrsto grafa (stolpčni ali črtni graf),
- ZZZS (Zavod za zdravstveno zavarovanje Slovenije) številko,
- tabelo, za katero želimo pregledati podatke (dnevni vnosi ali izmerjen sladkor).

Pri dnevnih vnosih stoplci prikazujejo hranilne vrednosti za točen datum.



Slika 6.12: Stran za prikaz grafov

Poglavje 7

Zaključki in nadaljnje delo

V diplomski nalogi smo najprej proučili različne tehnologije in se nato odločili za najprimernejše (*HTML, CSS, JavaScript* na odjemalskem delu, in *Laravel* ter seveda *PHP* na strežniškem delu). V prvem koraku sem Laravel projekt povezal s podatkovno bazo (*MySQL*) ter projekt postavil v oblak (*Heroku*) za testiranje pri razvoju. V naslednjih korakih sem ločil posamezne dele aplikacije, ker sem se uporabiljal *MVC* arhitekturo. Vmesnik *artisan* mi je olajšal porazdelitev aplikacije. Povezava s podatkovno bazo je bila narjena z nekaj kliki. Večjih težav pri implementaciji nisem imel. Najbolj zanimivo se mi je zdelo pravilno postaviti uporabniški vmesnik. Da bi si pomagal, sem pregledal nekatere že uporabljene spletne aplikacije v zdravstvu. Naredil sem tudi kratko raziskavo, in sicer tako, da sem določene zaposlene spraševal, kaj jim je pri uporabi spletnih aplikacij najbolj pomembno. Ugotovil sem, da je pomembna hitrost delovanja aplikacije in to, da uporabniški vmesnik ostane zelo preprost. Vse navedene vsebine sem upošteval pri izdelavi aplikacije.

Menim, da sem dosegel svoj cilj in naredil aplikacijo, ki bi zaposlenim olajšala delo v enoti intenzivne nege. Pri implementaciji sem se držal dobre prakse v izdelavi spletnih aplikacij, s pomočjo katerih je potem možno pri nadaljnem delu lahko implementirati dodatne funkcionalnosti.

7.1 Nadaljnje delo

Spletna aplikacija bi se lahko izboljšala tako, da bi se povezala z uvoznikom hrane. Potem ne bi bilo potrebno ročno vnašati informacij o hrani, ker bi sistem uvoznika samodejno sporočil hranilne snovi in energijsko vrednost za posamezno hrano. Ker je aplikacija narejena tako, da ima možnost izdelave *REST API*, je ena od možnosti za nadaljnje delo poveza z mobilno aplikacijo. Če pa ne bi želeli izdelati mobilne aplikacije, pa lahko spletno aplikacijo na mobilnih napravah prikažemo s pomočjo odzivnega dizajna (*ang. responsive design*).

Literatura

[1] Douglas Crockford. *JavaScript: The Good Parts: The Good Parts*. O'Reilly Media, Inc., 2008.

[2] Wei Cui, Lin Huang, LiJing Liang, and Jing Li. The research of php development framework based on mvc pattern. In *Computer Sciences and Convergence Information Technology, 2009. ICCIT'09. Fourth International Conference on*, pages 947–949. IEEE, 2009.

[3] Ibrahim Elmadfa and Claus Leitzmann. *Ernährung des Menschen*. Ulmer Stuttgart, 2004.

[4] enemon.si. Vitamin, may 2014.

[5] Laravel 5 Essentials. *Bean, Martin*. Packt Publishing Ltd, 2015.

[6] Brian P Hogan. *HTML5 und CSS3*. O'Reilly Germany, 2011.

[7] Shenja Jeworek. Was ist material design von google und was gibt es zu beachten., may 2014.

[8] Peter Kapele. Koliko beljakovin na dan je treba zaužiti, nov 2013.

[9] Peter Kapele. Maščobe v prehrani, aug 2015.

[10] Lekarnar.com. Aminokisline - osnovni gradniki v telesu, feb 2014.

[11] Jevšnik Mojca, Bauer Martin, Ovca Andrej, Ruža Pandel Mikuš, and Poljšak Borut. Skrb za zdravo in varno prehrano med starostniki. *Raziskovalni dan*, pages 25–28, nov 2009.

[12] Downie Nick. Chart.js documentation, apr 2014.

[13] Rashidah F Olanrewaju, Thouhedul Islam, and N Ali. An empirical study of the evolution of php mvc framework. In *Advanced Computer and Communication Engineering Technology*, pages 399–410. Springer, 2015.

[14] Taylor Otwell. Console commands, oct 2016.

[15] J Stein and A Jordan. Ernährung bei krankheiten des gastrointestinaltrakts. In *Praxishandbuch klinische Ernährung und Infusionstherapie*, pages 582–626. Springer, 2003.

[16] Jürgen Stein and K-W Jauch. *Praxishandbuch klinische Ernährung und Infusionstherapie*. Springer-Verlag, 2013.

[17] Frankič Tamara and Salobir Janez. Antioksidanti v prehrani živali: pomen za živali in porabnike. *V Zbornik predavanja*, pages 34–40, nov 2007.

[18] Thomas Theis. *Einstieg in PHP 5.5 und MySQL 5.6*. Galileo Press, 2013.